Sorting and searching cheatsheet for coding interviews

Introduction

Sorting is the act of rearranging elements in a sequence in order, either in numerical or lexicographical order, and either ascending or descending.
A number of basic algorithms run in $O(n^2)$ and should not be used in interviews. In algorithm interviews, you're unlikely to need to implement any of the sorting algorithms from scratch. Instead you would need to sort the input using your language's default sorting function so that you can use binary searches on them.
On a sorted array of elements, by leveraging on its sorted property, searching can be done on them in faster than O(n) time by using a binary search. Binary search compares the target value with the middle element of the array, which informs the algorithm whether the target value lies in the left half or the right half, and this comparison proceeds on the remaining half until the target is found or the remaining half is empty.

Learning resources

While you're unlikely to be asked to implement a sorting algorithm from scratch during an interview, it is good to know the various time complexities of the different sorting algorithms.

- Readings
    - Sorting Out The Basics Behind Sorting Algorithms, basecs
    - Binary Search, Khan Academy
- Additional (only if you have time)
    - Exponentially Easy Selection Sort, basecs
    - Bubbling Up With Bubble Sorts, basecs
    - Inching Towards Insertion Sort, basecs

    - Making Sense of Merge Sort (Part 1), basecs
    - Making Sense of Merge Sort (Part 2), basecs
    - Pivoting To Understand Quicksort (Part 1), basecs
    - Pivoting To Understand Quicksort (Part 2), basecs
    - Counting Linearly With Counting Sort, basecs
    - Getting To The Root Of Sorting With Radix Sort, basecs
- Videos
    - Heapsort (slides), Samuel Albanie, University of Cambridge
    - Quicksort (slides), Samuel Albanie, University of Cambridge
    - Lower bounds for comparison sorts (slides), Samuel Albanie, University of Cambridge
    - Counting sort (slides), Samuel Albanie, University of Cambridge
    - Radix sort (slides), Samuel Albanie, University of Cambridge
    - Bucket sort (slides), Samuel Albanie, University of Cambridge

Time complexity

Algorithm	Time	Space
Bubble sort	$O(n^2)$	O(1)
Insertion sort	$O(n^2)$	O(1)
Selection sort	$O(n^2)$	O(1)
Quicksort	O(nlog(n))	O(log(n))
Mergesort	O(nlog(n))	O(n)
Heapsort	O(nlog(n))	O(1)
Counting sort	O(n + k)	O(k)
Radix sort	O(nk)	O(n + k)

Algorithm	Big-O

Binary search	O(log(n))

Things to look out for during interviews

Make sure you know the time and space complexity of the language's default sorting algorithm! The time complexity is almost definitely O(nlog(n))). Bonus points if you can name the sort. In Python, it's Timsort. In Java, an implementation of Timsort is used for sorting objects, and Dual-Pivot Quicksort is used for sorting primitives.

Corner cases

- Empty sequence
- Sequence with one element
- Sequence with two elements
- Sequence containing duplicate elements.

Techniques

Sorted inputs

When a given sequence is in a sorted order (be it ascending or descending), using binary search should be one of the first things that come to your mind.

Sorting an input that has limited range

Counting sort is a non-comparison-based sort you can use on numbers where you know the range of values beforehand. Examples: H-Index

Essential questions

These are essential questions to practice if you're studying for this topic.

- Binary Search
- Search in Rotated Sorted Array

Recommended practice questions

These are recommended questions to practice after you have studied for the topic and have practiced the essential questions.

- Kth Smallest Element in a Sorted Matrix
- Search a 2D Matrix
- Kth Largest Element in an Array
- Find Minimum in Rotated Sorted Array
- Median of Two Sorted Arrays